The Declt User Manual

Documentation extractor from Common Lisp to Texinfo, Version 2.0 "Kathryn Janeway"

Didier Verna <didier@didierverna.net>

Table of Contents

Copying

1 Introduction

`Declt` (pronounce "dec'let") is a reference manual generator for Common Lisp libraries. It works by loading an ASDF system and introspecting its contents. The generated documentation contains the description of the system itself and its local dependencies (other systems in the same distribution): components (modules and files), packages and definitions found in those packages.

Exported and internal definitions are listed separately. This allows the reader to have a quick view on the library's public API. Within each section, definitions are sorted lexicographically.

In addition to ASDF system components and packages, `Declt` documents the following definitions: constants, special variables, symbol macros, macros, `setf` expanders, compiler macros, functions (including `setf` ones), generic functions and methods (including `setf` ones), method combinations, conditions, structures, classes and types.

The generated documentation includes every possible bit of information that introspecting can provide: documentation strings, lambda lists (including qualifiers and specializers where appropriate), slots (including type, allocation and initialization arguments), definition source file *etc.*

Every documented item provides a full set of cross-references to related items: ASDF component dependencies, parents and children, classes direct methods, super and subclasses, slot readers and writers, `setf` expanders access and update functions *etc.*

Finally, `Declt` produces exhaustive and multiple-entry indexes for every documented item.

Reference manuals are generated in Texinfo format (compatible, but not requiring Texinfo 5). From there it is possible to produce readable / printable output in info, HTML, PDF, DVI and PostScript with tools such as `makeinfo`, `texi2dvi` or `texi2pdf`.

The primary example of documentation generated by `Declt` is *The Declt Reference Manual*.

2 Installation

`Declt` is currently an SBCL only library. It is provided as an ASDF 3 system. See Section A.2 [Supported Platforms], page 15, for more information.

See `Declt`'s homepage for tarballs, Git repository and online documentation. `Declt` is also available via Quicklisp.

In order to install and load the bare Lisp library, unpack it somewhere in the ASDF 3 source registry and type this at the REPL:

```
(asdf:load-system :net.didierverna.declt)
```

In addition to the bare Lisp library, the `Declt` distribution offers documentation in the form of 2 different manuals: the User Manual (you are reading it) and the *Reference Manual*. The latter is generated by `Declt` itself. If you want to benefit from all those wonders, some bits of manual installation are needed. After unpacking somewhere in the ASDF 3 source registry, please perform the following steps, in order.

1. Edit `make/config.make` to your specific needs.
2. Type `make` to compile the documentation (user manual and reference manual). By default, the documentation is built in info, PDF and HTML formats. If you want other formats (DVI and PostScript are available), type `make all-formats`. You can also type individually `make dvi` and/or `make ps` in order to get the corresponding format.
3. Type `make install` to install the documentation. If you have compiled the documentation in DVI and PostScript format, those will be installed as well.

Type `make uninstall` to uninstall the library.

3 Usage

`Declt` itself resides in a package called `net.didierverna.declt`. You can automatically nickname this package with the following function.

`nickname-package` [*NICKNAME*] [Function]
Add NICKNAME (:DECLT by default) to the :NET.DIDIERVERNA.DECLT package.

Apart from that, `Declt` has a single entry point: the `declt` function.

`declt` *SYSTEM-NAME :KEY VALUE...* [Function]
Generate a reference manual in Texinfo format for ASDF SYSTEM-NAME.

3.1 Customization

The `Declt` function accepts a number of keyword arguments allowing you to customize the generated reference manual. They are listed below.

`:library-name`
The library name, used in the reference manual title and at some other places. It defaults to the system name, but you are encouraged to provide a more human-readable version, such as `"Declt"` instead of just `net.didierverna.declt`.

`:texi-file`
The full path of the generated Texinfo file. It defaults to `LIBRARY-NAME.texi` in the current directory.

`:info-file`
The base name of the info file, sans extension. The default is built from `TEXI-FILE`.

`:tagline` A tagline for the library, use in the reference manual subtitle. It defaults to the system's long name or description.

`:version` The library's version, used in the reference manual subtitle and at some other places. It defaults to the system's version.

`:maintainer`
The library's maintainer. It defaults to the system's maintainer or author. Note that having a maintainer is mandatory to produce a reference manual (the maintainer appear on the title page).

`:contact` The contact address for the library. The default is extracted from the system definition (either the `maintainer`, the `author` or the `mailto` slots).

`:license` The library's license type. This information is used to insert licensing information at several places in the manual. The default is `nil` and the possible values are: `:mit`, `:bsd`, `:gpl` and `:lgpl`. The corresponding license texts are stored in the **licenses** parameter. Please ask if you need other kinds of licenses added to `Declt`. Note that this information is **not** extracted from the system's `license` slot, as this slot is not well defined.

`:copyright-date`
A copyright date to be used in conjunction with the licensing information. It defaults to the current year. You may use any kind of string here, such as `"2013"`, `"2010, 2011"`, `"2010--2013"` *etc.*

`:declt-notice`
Controls the output of a small paragraph about automatic manual generation by Declt. Possible values are `nil`, `:short` and `:long` (the default). I would be grateful

if you kept at least the short version in your manuals, as an acknowledgment for using `Declt`.

`:introduction`
: A potential contents for an introductive chapter.

`:conclusion`
: A potential contents for a conclusive chapter.

`:hyperlinks`
: Whether to create hyperlinks to files or directories in the reference manual. Note that those links being specific to the machine on which the manual was generated, it is preferable to keep it to `nil` for creating reference manuals meant to be put online.

Note that both the introduction and the conclusion may contain Texinfo directives (no post-processing will occur). All other textual material is considered raw text and will be properly escaped for Texinfo.

3.2 Coding Style

Some elements of your own coding style will affect the reference manuals generated by `Declt`. This section provides some recommendations that will make the generated output look nicer.

3.2.1 Taglines

Unless you provide it with an explicit `:tagline` argument, `declt` uses the system's long name or description to construct a subtitle. Consequently, it is advisable to use a single (short) line of text for these slots. The system's long name should typically be the expansion of the system's name, if that's an acronym, or be left to `nil`.

3.2.2 Docstrings

`Declt` tries to make the generated output look nicer in various ways. For example, `setf` functions are documented right after the corresponding reader (if any) instead of being listed under the "S" letter. In a similar vein, methods are documented as elements of their respective generic function, not as toplevel definitions.

One thing that you can influence is `Declt`'s attempt at concatenating definitions. Concatenating may occur when there are definitions for both `symbol` and `(setf symbol)`. This happens for accessor functions, generic functions or `setf` expanders. This also happens for accessor methods. If possible, `Declt` will try to generate a *single* definition for both the reader and the writer. That is only possible, however, if both definitions would render the same documentation, *i.e.* same package, source file and docstring.

If you don't provide a docstring, concatenation will work. If you provide different docstrings (like "Set the value of . . ." and "Get the value of . . ."), you will effectively prevent concatenation from happening. One thing I like to do is to provide the *same* neutral docstring for readers and writers. For instance "Access the value of . . .". This way, definitions can both provide a docstring and be concatenated together.

3.2.3 80 Columns

All text coming from either Common Lisp or one of `declt`'s initialization arguments (`:introduction` and `:conclusion` excepted) is properly escaped for the Texinfo format, so you don't need to worry about that. `Declt` also attempts to do some DWIM processing on things like docstrings, system long description *etc.*. In particular, it will try to detect paragraphs and short lines that should probably stand on their own. In order to do that, it assumes that your plain text is formatted to fit 80 columns.

3.3 Caveats

`Declt` currently has two main limitations that you need to understand in order to avoid bad surprises.

3.3.1 SBCL Only

First, `Declt` is an SBCL-only library. That is because it relies on `sb-introspect`. This limitation may be lifted in the future by using equivalent API from other Common Lisp implementations, but in the meantime, this means two things.

1. `Declt` can only document libraries that work with SBCL, because it needs to load them (see Chapter 1 [Introduction], page 3).
2. If your ASDF system contains vendor-specific modules or components, `Declt` will only be able to document SBCL-specific ones.

Note that more generally, `Declt` only documents modules or components that ASDF actually loads, so if your system definition contains some form of conditional inclusion, this will affect the generated documentation.

3.3.2 Foreign Definitions

In order to understand what a *foreign definition* is, you need to understand how `Declt` works first. This was briefly explained in the introduction, but here is the story again: `Declt` first loads the ASDF system from which it retrieves modules and components, notably including Lisp files. It then scans those Lisp files (including the system file itself), looking for package definitions (toplevel calls to `DEFPACKAGE` forms). Finally, it collects all definitions for the symbols in those packages.

That is the extent of what is documented by `Declt`. In this context, a *foreign definition* is a definition that `Declt` would miss because of the way it works. This includes definitions found in one of the system's files, but belonging to a package that was not defined in the system, packages defined by non-toplevel forms *etc.*

`Declt` currently does not document foreign definitions, although in some cases, it is already possible to avoid missing them. In fact, the very question of deciding on whether foreign definitions should actually be documented is still open for discussion. See the `TODO` file in the distribution for more information.

3.3.3 Method Combinations

The method combination interface in Common Lisp is a bit weird. In particular, although you define method combinations globally, changing them afterwards won't affect already created generic functions. As a result, you could in theory end up with **many** different method combinations with the same name, used in various generic functions. See this blog entry (`http://www.didierverna.net/blog/index.php?post/2013/08/16/Lisp-Corner-Cases%3A-Method-Combinations`) for more explanations.

`Declt` assumes however that you have some sanity and only define method combinations once per name. They are documented as top level items and generic functions using them provide cross-references.

4 Advanced Usage

This section contains information about different features that are present in `Declt` because of design decisions, but that I expect to be used only rarely, if at all.

4.1 Version Numbering

As `Declt` evolves over time, you might one day feel the need for conditionalizing your code on the version of the library.

The first thing you can do to access the current version number of `Declt` is use the `version` function.

`version` **&optional** (*TYPE* **:number**) [Function]

Return the current version number of `Declt`. *TYPE* can be one of `:number`, `:short` or `:long`. For `:number`, the returned value is a fixnum. Otherwise, it is a string.

A `Declt` version is characterized by 4 elements as described below.

- A major version number stored in the parameter `*release-major-level*`.
- A minor version number, stored in the parameter `*release-minor-level*`.
- A release status stored in the parameter `*release-status*`. The status of a release can be `:alpha`, `:beta`, `:rc` (standing for "release candidate") or `:patchlevel`. These are in effect 4 levels of expected stability.
- A status-specific version number stored in the parameter `*release-status-level*`. Status levels start at 1 (alpha 1, beta 1 and release candidate 1) except for stable versions, in which case patch levels start at 0 (*e.g.* 2.4.0).

In addition to that, each version of `Declt` (in the sense *major.minor*, regardless of the status) has a name, stored in the parameter `*release-name*`. The general theme for `Declt` is "Star Trek Characters".

Here is how the `version` function computes its value.

- A version `:number` is computed as *major* `*` *10000* `+` *minor* `*` *100* `+` *patchlevel*, effectively leaving two digits for each level. Note that alpha, beta and release candidate status are ignored in version numbers (this is as if the corresponding status level was considered to be always 0). Only stable releases have their level taken into account.
- A `:short` version will appear like this for unstable releases: 1.3a4, 2.5b8 or 4.2rc1. Remember that alpha, beta or release candidate levels start at 1. Patchlevels for stable releases start at 0 but 0 is ignored in the output. So for instance, version 4.3.2 will appear as-is, while version 1.3.0 will appear as just 1.3.
- A `:long` version is expanded from the short one, and includes the release name. For instance, 1.3 alpha 4 `"Uhura"`, 2.5 beta 8 `"Scotty"`, 4.2 release candidate 1 `"Spock"` or 4.3.2 `"Counsellor Troy"`. As for the short version, a patchlevel of 0 is ignored in the output: 1.3 `"Uhura"`.

5 Conclusion

So that's it I guess. You know all about `Declt` now. The next step is to polish your own libraries so that `Declt` can extract meaningful documentation from them.

Then, you will want to run `Declt` on all the other libraries you use, in order to finally know how they work.

Now, go my friend. Go document the whole Lisp world!

Appendix A Technical Notes

This chapter contains important information about the library's configuration and portability concerns.

A.1 Configuration

Some aspects of `Declt`'s behavior can be configured *before* the library system is actually loaded. `Declt` stores its user-level configuration (along with some other setup parameters) in another ASDF system called '`net.didierverna.declt.setup`' (and the eponym package). In order to configure the library (I repeat, prior to loading it), you will typically do something like this:

```
(require "asdf")
(asdf:load-system :net.didierverna.declt.setup)
(net.didierverna.declt.setup:configure <option> <value>)
```

`configure` *KEY VALUE* [Function]
Set KEY to VALUE in the current Declt configuration.

Out of curiosity, you can also inquire the current configuration for specific options with the following function.

`configuration` *KEY* [Function]
Return KEY's value in the current Declt configuration.

Currently, the following options are provided.

`:swank-eval-in-emacs`
This option is only useful if you use Slime, and mostly if you plan on hacking `Declt` itself. The library provides indentation information for some of its functions directly embedded in the code. This information can be automatically transmitted to (X)Emacs when the ASDF system is loaded if you set this option to `t`. However, note that for this to work, the Slime variable `slime-enable-evaluate-in-emacs` must also be set to `t` in your (X)Emacs session. If you're interested to know how this process works, I have described it in the following blog entry: `http://www.didierverna.net/blog/index.php?post/2011/07/20/One-more-indentation-hack`.

A.2 Supported Platforms

`Declt` is an ASDF 3 library. It currently works on Unix (including MacOS X) and Windows (Cygwin or MinGW) with SBCL only. Other Lisp implementations are not currently supported, as `Declt` relies on SBCL's `sb-introspect` contrib.

Appendix B Indexes

B.1 Concepts

B.2 Functions

C

D

N

V

B.3 Variables

*

N

P

S

B.4 Data Types

N

P

S